A Preprocessing for Approximate String Matching

Baba, Kensuke
Kyushu University

Nakatoh, Tetsuya
Kyushu University

Yamada, Yasuhiro

Ikeda, Daisuke
Kyushu University

https://hdl.handle.net/2324/20275

出版情報：Informatics Engineering and Information Science, 2011-11. Springer-Verlag
バージョン：
権利関係：

A Preprocessing for Approximate String Matching[*]

Kensuke Baba[†] Tetsuya Nakatoh Yasuhiro Yamada Daisuke Ikeda

Abstract

Approximate string matching is a basic and important concept in many applications of information retrieval. This paper proposes an algorithm for the problem of approximate string matching. The algorithm solves the match-count problem as a preprocessing. For input strings of each length n, the time complexities of the approximate string matching problem and the match-count problem are $O(n^2)$ and $O(n \log n)$, respectively. Therefore, the computation time of the algorithm is expected to be short when the scope of search is drastically restricted by the preprocessing. This paper makes clear the relation between the solutions of the two problems.

Keywords: Algorithm, approximate string matching, FFT.

1 Introduction

Similarity on strings is one of the most important concepts in many applications of information retrieval. Especially for mining on a huge database such as homology search in biology, the process of pattern matching is required to be fast and flexible.

The aim of this paper is to speed up the process of approximate string matching [7]. Exact string matching is the problem to find the occurrences of a (short) string, called a "pattern", in another (long) string, called a "text". The problem of approximate string matching is defined to be the string matching problem which allows some errors with a threshold based on the edit distance [11]. The edit distance is the minimal number of the edit operations which transform one string to the other, where the permitted operations are "insertion", "deletion", and "replacement" of a character. The generalizations in the sense of weight [11] and local similarity [10] are the essence of some popular systems for sequence analysis in biology [9]. It is significant for many applications to speedup solving the approximate string matching problem.

An approach to the speedup is parallel computation which depends on the performance of a computer. If we assume a computational model which corresponds to a computer with a multi-core processor, a straightforward method is to part the text with overlaps, and then solve the problem for each parted text separately. Another simple parallel computation is "wavefront" which computes the matrix for the dynamic programming approach. Myers [7] proposed an efficient algorithm based on the idea of "bit-parallel [8]" for the approximate string matching problem. The speedup method of this paper is to compute another problem which can be solved faster than the original problem as a preprocessing, that is, the proposed method can be applied with the previous speedup methods simultaneously.

In this paper, the match-count problem [6] is considered as the preprocessing. The problem allows only replacement as the edit operation for the idea of distance. Although this approach makes no improvement of the computation time in the worst case, there exist significant speedup methods for this problem and a practical speedup is expected in some applications in which the occurrences of the pattern are not so many. While the time complexity of the standard algorithm for the approximate string matching problem is $O(n^2)$ for input strings of length n, the match-count problem is solved by the first Fourier transformation (FFT) in $O(n \log n)$ [5, 6], and moreover, some improvements for the computation time were proposed [1, 3, 2]. This paper makes clear the relation between the solutions of the approximate string matching problem and the match-count problem, and proposes an algorithm which solves the match-count problem as a preprocessing for solving the approximate string matching problem.

2 Formalization

Let Σ be a finite set of characters. For an integer $n > 0$, Σ^n denotes the set of the strings of length n over Σ. Σ^* denotes the set of the strings of finite length over Σ and by ε the empty string. For a string u, $|u|$ denotes the length of u and u_i denotes the ith element of u for $1 \leq i \leq |u|$. The string $u_i u_{i+1} \cdots u_j$ is a *substring* of u, and denoted by $u_{i,j}$. In particular, $u_{i,j}$ is called a *prefix* if $i = 1$ and a *suffix* if $j = |u|$.

[*] An edited version of this report was published in: *Informatics Engineering and Information Science, Communications in Computer and Information Science, vol. 252, pp. 610–615, Springer-Verlag, Nov, 2011.*

[†] Research and Development Division, Kyushu University Library, *baba@lib.kyushu-u.ac.jp*

Let $Aa = \{ua \mid u \in A\}$ for $A \subseteq \Sigma^*$ and $a \in \Sigma$. An *edit transcript* from $u \in \Sigma^*$ to $v \in \Sigma^*$ is a string on $\{I, D, R, M\}$, such that, the set $T(u, v)$ of the edit transcripts from u to v is

- $T(u, v) = \{\varepsilon\}$ if $uv = \varepsilon$;
- $T(u, v) = T(u, v')I$ if $u = \varepsilon$ and $v = v'a$ for $a \in \Sigma$;
- $T(u, v) = T(u', v)D$ if $u = u'a$ and $v = \varepsilon$ for $a \in \Sigma$;
- $T(u, v) = T(u, v')I + T(u', v)D + T(u', v')R$ if $u = u'a$, $v = v'b$, and $a \neq b$ for $a, b \in \Sigma$;
- $T(u, v) = T(u, v')I + T(u', v)D + T(u', v')M$ if $u = u'a$, $v = v'b$, and $a = b$ for $a, b \in \Sigma$.

An edit transcript from u to v is *optimum* if the number of occurrences of I, D, and R in the edit transcript is minimum in $T(u, v)$. The *edit distance* $d(u, v)$ between u and v is the number of occurrences of I, D, and R in an optimum edit transcript.

Definition 1 *For $p, t \in \Sigma^*$ and an integer ℓ, the* approximate string matching problem *is to find the substrings t' of t, such that, $d(p, t') \leq \ell$.*

The *score vector* $S(p, t)$ between $p \in \Sigma^m$ and $t \in \Sigma^n$ (assume $m < n$) is the vector whose ith element s_i is the number of matches between p and the substring $t_{i,i+m-1}$ of t for $1 \leq i \leq n-m+1$. Let δ be a function from $\Sigma \times \Sigma$ to $\{0, 1\}$, such that, for $a, b \in \Sigma$, $\delta(a, b)$ is 1 if $a = b$, and 0 otherwise. Then, for $1 \leq i \leq n-m+1$, the ith element of the score vector is

$$s_i = \sum_{j=1}^{m} \delta(p_j, t_{i+j-1}). \qquad (1)$$

Definition 2 *For $p, t \in \Sigma^*$, the* match-count problem *is to compute $S(p, t)$.*

3 Standard Algorithms

3.1 Approximate String Matching Problem

The edit distance between $u \in \Sigma^m$ and $v \in \Sigma^n$ is computed in $O(mn)$ time by the dynamic programming approach [11]. In this approach, the *cost matrix* $C(u, v)$ is evaluated, whose (i, j)-element $c_{i,j}$ is the edit distance between the prefix $u_{1,i}$ of u and the prefix $v_{1,j}$ of v. By the definition of the edit distance,

$$c_{i,j} = \min\{c_{i-1,j-1} + 1 - \delta(u_i, v_j), c_{i-1,j} + 1, c_{i,j-1} + 1\}$$

for $1 \leq i \leq m$ and $1 \leq j \leq n$. The base conditions are $c_{i,0} = i$ and $c_{0,j} = j$. The (m, n)-element of the cost matrix is the edit distance between u and v and obtained by computing the $m \times n$ elements of the cost matrix.

The approximate string matching problem is solved also in $O(mn)$ time by the previous approach on the base conditions $c_{i,0} = i$ and $c_{0,j} = 0$, which is clear from the idea of the Smith-Waterman algorithm [10]. $c'_{i,j}$ denotes the (i, j)-element of the cost matrix on these base conditions.

In the strict sense, the previous method finds the positions which the target substrings start (or end), and moreover there can exist more than two target substrings which start at one position. These problems are solved by a linear-time operation called a "traceback [6]". In the rest of this paper, we focus on finding the positions of the target substrings in the approximate string matching problem.

3.2 Match-count Problem

A naive method for the match-count problem is, for $p \in \Sigma^m$ and $t \in \Sigma^n$, to make the $m \times n$ matrix $D(p, t)$ whose (i, j)-element is $\delta(p_i, t_j)$ and compute each s_k by Eq. 1 for $1 \leq k \leq n - m + 1$. Therefore, $S(p, t)$ is obtained by $m \times (n-m+1)$ comparisons and $(m-1) \times (n-m+1)$ add operations. Thus, the time complexity of this naive algorithm is $O(mn)$.

The most straightforward method of parallel computation for the match-count problem is to part t or p into substrings. Intuitively, in this method, using k computers (processors, or cores) yields a k-times speedup. Clearly, by $t_{i,j}$ and p, $C(t, p)$ is obtained from the ith element to the $(j - m + 1)$th element. Therefore, by parting t into k substrings with overlaps of length $m - 1$ and combining the results, $C(t, p)$ is obtained by k distinct computations. If p is parted, the following is clear in general. Let c_i^p be the ith element of the score vector $C(t, p)$ and c_i^q the ith element of $C(t, q)$. Then, the ith element of $C(t, pq)$ is $c_i^p + c_{m+i}^q$, where m is the length of p. Therefore, we can also expect straightforward speedup except for the overhead.

Additionally, for the match-count problem, there exists an efficient algorithm using the fast Fourier transform (FFT) [6]. The convolution

$$w_i = \sum_{j=1}^{m} u_j \cdot v_{i-j} \quad (1 \leq i \leq m)$$

of two m-dimensional vectors u and v can be computed in $O(m \log m)$ time by FFT [4]. Therefore, the score vector between two strings each of length m is computed in $O(m \log m)$ time. By parting t into overlapping substrings and padding p with a never-match character, we obtain an $O(n \log m)$ algorithm.

4 Relation between Score Vector and Edit Distance

We make clear the relation between the score vector and the edit distance, and then propose an algorithm for the approximate string matching problem using the result of the match-count problem for a speedup.

We consider the score vector between $p \in \Sigma^m$ and $t \in \Sigma^n$ and the edit distances between p and the substrings of t. Now we extend the definition of the score vector. For $k \leq 0$ and $n+1 \leq k$, we assume that t_k is a never-match character, that is, $\delta(p_j, t_k) = 0$ for any $1 \leq j \leq m$. Then, s_i in Eq. 1 is extended for $i \leq 0$ and $n - m + 1 \leq i$.

Lemma 1 *If there exists a pair of i and j such that $d(p, t_{i,j}) \leq \ell$, then there exists r such that $\sum_{k=r}^{r+\ell} s_k \geq m - \ell$ and $i - \ell \leq r \leq i$.*

Proof. Let $g = |t_{i,j}| - |p|$. By the definition of the edit distance, if $d(p, t_{i,j}) \leq \ell$, then $|g| \leq \ell$ and there exist at least $m - \ell$ matches. Therefore, $d(p, t_{i,j}) \leq \ell$ implies

$$\sum_{k=i-\lfloor(\ell-g)/2\rfloor}^{i+\lfloor(\ell+g)/2\rfloor} s_k \geq m - \ell.$$

Since $\lfloor(\ell+g)/2\rfloor \leq \ell$ and $\lfloor(\ell-g)/2\rfloor \leq \ell$ by $-\ell \leq g \leq \ell$, we have only to consider the summation $\sum_{k=r}^{r+\ell} s_k$ for $i - \ell \leq r \leq i$.

By the previous lemma, if $\sum_{k=r}^{r+\ell} s_k < m - \ell$ for $i - \ell \leq r \leq i$, then there is no pair of i and j such that $d(p, t_{i,j}) \leq \ell$. That is, the candidates of the approximate string matching problem is reduced by the result of the match-count problem. The outline of an algorithm based on this idea is the following.

Algorithm A:
Input: $p \in \Sigma^m$, $t \in \Sigma^n$, ℓ
Output: $P = \{i \mid 1 \leq i \leq n, \exists k.d(p, t_{k,i}) \leq \ell\}$

```
for $(2 - m \leq i \leq n)$ compute $s_i$ ;

$R := 0$ ; $Q := \emptyset$ ;
for $(3 - m - \ell \leq i \leq 1 - m)$ $s_i := 0$ ;
for $(2 - m \leq i \leq n)$ {
    $R := R + s_i - s_{i-\ell+1}$ ;
    if $(R \geq m - \ell)$
        $Q := Q \cup \{i - \ell + 1, i - \ell + 2, \ldots, i\}$ ;
}

for $(i \in Q)$ compute $c'_{m,i}$ and find $P$.
```

In the algorithm, s_i and $c'_{m,i}$ are computed by the standard algorithms for the match-count problem and the approximate string matching problem, respectively. Then, the following theorem is clear by Lemma 1.

Theorem 1 *Algorithm A solves the problem of approximate string matching.*

Intuitively, Q in Algorithm A is the set of the positions which can be the target of approximate string matching after the screening by match-count. Therefore, the algorithm is effective when the size of Q is extremely small compared with n. In the worst case, the number of positions is not decreased by the preprocessing, hence the time complexity of the algorithm is $O(mn)$.

5 Conclusion

An algorithm for the approximate string matching problem which uses the result of the match-count problem as a preprocessing was proposed. We made clear the relation between the results of the two problems, and thereby constructed the algorithm. The computation time of the algorithm is expected to be short in the case where the number of the occurrences of the pattern is small compared to the length of the text.

References

[1] M. J. Atallah, F. Chyzak, and P. Dumas. A randomized algorithm for approximate string matching. *Algorithmica*, 29:468–486, 2001.

[2] K. Baba. String matching with mismatches by real-valued FFT. In *Computational Science and Its Applications - ICCSA 2010, Part IV*, volume 6019 of *Lecture Notes in Computer Science*, pages 273–283, 2010.

[3] K. Baba, A. Shinohara, M. Takeda, S. Inenaga, and S. Arikawa. A note on randomized algorithm for string matching with mismatches. *Nordic Journal of Computing*, 10(1):2–12, 2003.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms, Second Edition.* MIT Press, 2001.

[5] M. J. Fischer and M. S. Paterson. String-matching and other products. In *Complexity of Computation (Proceedings of the SIAM-AMS Applied Mathematics Symposium, New York, 1973)*, pages 113–125, 1974.

[6] D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.

[7] G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, 46(3):395–415, 1999.

[8] G. Navarro. A guided tuor to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[9] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. In *Proc. Natl. Acad. Sci. USA*, volume 85, pages 2444–2448, 1988.

[10] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[11] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.